4 Second-Order Optimization Techniques

In this chapter we describe fundamental optimization algorithms that leverage the first and second derivatives, or likewise the *gradient* and *Hessian* of a function. These techniques, collectively called *second-order optimization methods*, are popular in particular applications of machine learning today. In analogy to the previous chapter, here we begin with a discussion of the second-order optimality condition. We then discuss quadratic functions as well as the notion of curvature defined by second derivatives, and the second-order Taylor series expansion. By exploiting a function's first- and second-order derivative information we can construct powerful local optimization methods, including the popular *Newton's method* and its extensions (commonly referred to as *Hessian-free* optimizers).

4.1 The Second-Order Optimality Condition

When discussing convexity/concavity of general mathematical functions we often talk about convexity/concavity *at a point*. To determine whether a general single-input function $g(w)$ is *convex* or *concave* at a point v, we check its curvature or second derivative information at that point (assuming it is at least twice-differentiable there): if $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v) \geq 0$ (or ≤ 0) then g is said to be convex (or concave) at v.

An analogous statement can be made for a function g with multi-dimensional input: if the Hessian matrix evaluated at a point $\mathbf{v}$, denoted by $\nabla^2 g(\mathbf{v})$, has all nonnegative (or nonpositive) *eigenvalues* then g is said to be convex (or concave) at $\mathbf{v}$, in which case the Hessian matrix itself is called positive (or negative) semi-definite.

Based on these point-wise convexity/concavity definitions, the function $g(w)$ is said to be convex *everywhere* if its second derivative $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w)$ is always nonnegative. Likewise $g(\mathbf{w})$ is convex *everywhere* if $\nabla^2 g(\mathbf{w})$ always has nonnegative eigenvalues. This is generally referred to as the *second-order definition of convexity*.

Example 4.1 Convexity of single-input functions

In this example we use the second-order definition of convexity to verify whether each of the functions shown in Figure 4.1 is convex or not.

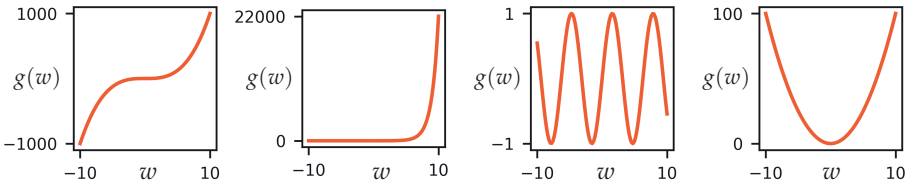

Figure 4.1 Figure associated with Example 4.1. From left to right, plots of functions $g(w) = w^3$, $g(w) = e^w$, $g(w) = \sin(w)$, and $g(w) = w^2$.

- $g(w) = w^3$ has second derivative $\frac{\mathrm{d}^2}{\mathrm{d}w^2} g(w) = 6w$, which is not always nonnegative, hence g is not convex.
- $g(w) = e^w$ has second derivative $\frac{\mathrm{d}^2}{\mathrm{d}w^2} g(w) = e^w$, which is positive for any choice of w, and g is therefore convex.
- $g(w) = \sin(w)$ has second derivative $\frac{\mathrm{d}^2}{\mathrm{d}w^2} g(w) = -\sin(w)$. Since this is not always nonnegative, g is nonconvex.
- $g(w) = w^2$ has second derivative $\frac{\mathrm{d}^2}{\mathrm{d}w^2} g(w) = 2$, and g is therefore convex.

Example 4.2 Convexity of multi-input quadratic functions
The multi-input quadratic function

$$g(\mathbf{w}) = a + \mathbf{b}^T\mathbf{w} + \mathbf{w}^T\mathbf{C}\,\mathbf{w} \tag{4.1}$$

has the Hessian matrix $\nabla^2 g(\mathbf{w}) = 2\mathbf{C}$ (assuming $\mathbf{C}$ is symmetric). Therefore its convexity is determined by studying the eigenvalues of $\mathbf{C}$.

By studying a few simple examples it is easy to come to some far-reaching conclusions about how the second derivative helps unveil the identity of stationary points. In Figure 4.2 we plot the three single-input functions we studied in Example 3.1 (defined in Equation (3.4)), along with their first- and second-order derivatives (shown in the top, middle, and bottom rows of the figure, respectively). In the top panels we mark the evaluation of all stationary points by the function in green (where we also show the tangent line in green). The corresponding evaluations by the first and second derivatives are marked in green as well in the middle and bottom panels of the figure, respectively.

By studying these simple examples in Figure 4.2 we can see consistent behavior of certain stationary points. In particular we can see consistency in how the value of a function's second derivative at a stationary point v helps us identify whether it is a local minimum, local maximum, or saddle point. A stationary point v is:

- a local (or global) minimum if $\frac{\mathrm{d}^2}{\mathrm{d}w^2} g(v) > 0$ (since it occurs at *convex* portions of a function),

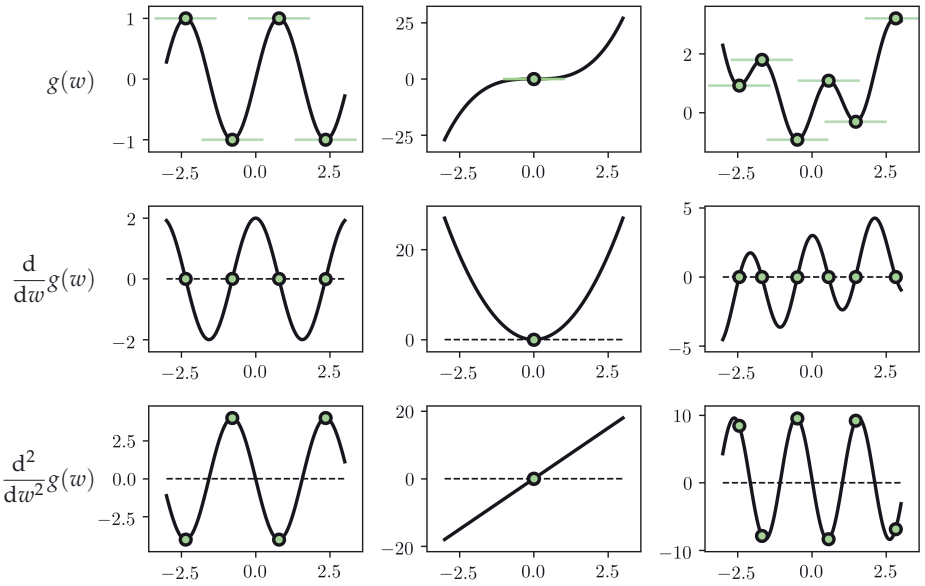

Figure 4.2 Three single-input functions along with their first- and second-order derivatives shown in the top, middle, and bottom panels, respectively. See text for further details.

- a local (or global) maximum if $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v) < 0$ (since it occurs at *concave* portions of a function),
- a saddle point if $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v) = 0$ and $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w)$ changes sign at $w = v$ (since it occurs at an *inflection point* of a function, i.e., where a function goes from concave to convex or vice versa).

These second-order characteristics hold more generally for any single-input function, and taken together form the *second-order condition for optimality* for single-input functions.

With multi-input functions the analogous second-order condition holds. As with all things having to do with convexity/concavity and the second-order derivative matrix (i.e., the Hessian), the second-order optimality condition for multi-input functions translates to the *eigenvalues* of the Hessian. More specifically, a stationary point $\mathbf{v}$ of a multi-input function $g(\mathbf{w})$ is:

- a local (or global) minimum if all eigenvalues of $\nabla^2 g(\mathbf{v})$ are positive (since it occurs at *convex* portions of a function),
- a local (or global) maximum if all eigenvalues of $\nabla^2 g(\mathbf{v})$ are negative (since it occurs at *concave* portions of a function),
- a saddle point if the eigenvalues of $\nabla^2 g(\mathbf{v})$ are of mixed values, i.e., some negative and some positive (since it occurs at an *inflection point* of a function).

Notice when the input dimension N equals 1, these rules reduce to those stated

for single-input functions as the Hessian matrix collapses into a single second-order derivative.

4.2 The Geometry of Second-Order Taylor Series

As we will see throughout this chapter, quadratic functions naturally arise when studying second-order optimization methods. In this section we first discuss quadratic functions with an emphasis on how we determine their overall shape, and whether they are *convex*, *concave*, or have a more complicated geometry. We then study quadratic functions generated by the second-order Taylor series approximation (see Appendix Section B.9 for a review of this concept), and in particular how these fundamental quadratics inherently describe the *local* curvature of a twice-differentiable function.

4.2.1 The general shape of single-input quadratic functions

The basic formula for a quadratic function with a single input takes the familiar form

$$g(w) = a + b\,w + c\,w^2 \tag{4.2}$$

where a, b, and c are all constant values controlling the shape of the function. In particular, the constant c controls the *convexity* or *concavity* of the function or, in other words, whether the quadratic faces upwards or downwards. When the value of c is *nonnegative* the quadratic function is *convex* and points *upwards*, regardless of how the other parameters are set. Conversely, when the value of c is *nonpositive* the quadratic is *concave* and points *downwards*. When $c = 0$ the quadratic reduces to a linear function (which can be considered both convex and concave).

In the left column of Figure 4.3 we plot two simple quadratics: the convex quadratic $g(w) = 6w^2$ (top-left panel) and the concave quadratic $g(w) = -w^2$ (bottom-left panel) to illustrate how the value of c controls the shape and convexity of the general quadratic function.

4.2.2 The general shape of multi-input quadratic functions

The multi-input quadratic function takes a form that is completely generalized from the single-input case, which we write as

$$g(\mathbf{w}) = a + \mathbf{b}^T\mathbf{w} + \mathbf{w}^T\mathbf{C}\,\mathbf{w} \tag{4.3}$$

where the input $\mathbf{w}$ is N-dimensional, a remains a constant, $\mathbf{b}$ is an $N \times 1$ vector, and $\mathbf{C}$ an $N \times N$ matrix (which we assume is *symmetric* for our purposes). Because

this quadratic is defined along many dimensions it can take on a wider variety of shapes than its single-input analog. For example, it can be convex along certain input dimensions and concave along others.

The generalization of the single-input test for convexity/concavity is no longer whether or not the *values* of $\mathbf{C}$ are positive or negative, but whether its *eigenvalues* are (see Appendix Section C.4.3). If the eigenvalues of the matrix are *all nonnegative* the quadratic is *convex*, if *all nonpositive* it is *concave*, if all equal zero it reduces to a linear function that is both convex and concave, and otherwise (i.e., if some of its eigenvalues are positive and others negative) it is neither convex nor concave.

In the middle and right columns of Figure 4.3 we show several examples of multi-input quadratic functions with $N = 2$ inputs. In all examples we have set a and $\mathbf{b}$ to zero and simply change the values of $\mathbf{C}$. For simplicity in all four cases $\mathbf{C}$ is chosen to be a diagonal matrix so that its eigenvalues are conveniently the entries on its diagonal.

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad \text{(top-middle panel of Figure 4.3)}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \text{(top-right panel of Figure 4.3)}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix} \qquad \text{(bottom-middle panel of Figure 4.3)}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \text{(bottom-right panel of Figure 4.3)}$$

4.2.3 Local curvature and the second-order Taylor series

Another way to think about the local convexity or concavity of a function g at a point v is via its second-order Taylor series approximation at that point (see Section B.9). This fundamental approximation taking the form

$$h(w) = g(v) + \left(\frac{\mathrm{d}}{\mathrm{d}w}g(v)\right)(w - v) + \frac{1}{2}\left(\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)\right)(w - v)^2 \tag{4.4}$$

is a true quadratic built using the (first and) second derivative of the function. Not only does the second-order approximation match the curvature of the underlying function at each point v in the function's domain, but if the function is convex at that point (due to its second derivative being nonnegative) then the second-order Taylor series is *convex everywhere*. Likewise if the function is concave at v, this approximating quadratic is *concave everywhere*.

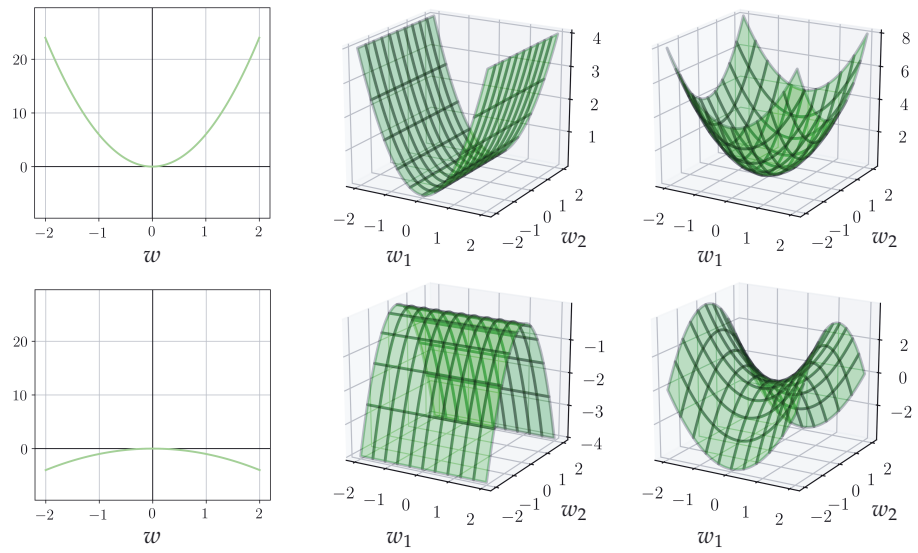

Figure 4.3 (top-left panel) A convex single-input quadratic. (bottom-left panel) a concave single-input quadratic. (top-middle and top-right panels) Two convex two-input quadratics. (bottom-middle panel) A concave two-input quadratic. (bottom-right panel) A two-input quadratic that is neither convex nor concave.

This concept holds analogously for multi-input functions as well. The second-order Taylor series approximation to a function taking in N-dimensional input at a point $\mathbf{v}$ is given by

$$h(\mathbf{w}) = g(\mathbf{v}) + \nabla g(\mathbf{v})^T(\mathbf{w} - \mathbf{v}) + \frac{1}{2}(\mathbf{w} - \mathbf{v})^T\nabla^2 g\,(\mathbf{v})\,(\mathbf{w} - \mathbf{v}). \tag{4.5}$$

Again, when the function g is convex at the point $\mathbf{v}$ the corresponding quadratic function is convex everywhere. Similar statements can be made when the function g is concave at the point $\mathbf{v}$, or neither convex or concave there.

Example 4.3 Local convexity/concavity and the second-order Taylor series
In Figure 4.4 we show the function

$$g(w) = \sin(3w) + 0.1w^2 \tag{4.6}$$

drawn in black, along with the second-order Taylor series quadratic approximation shown in turquoise at three example points (one per panel). We can see in this figure that the local convexity/concavity of the function is perfectly reflected in the shape of the associated quadratic approximation. That is, at points of local convexity (as in the first and second panel of the figure) the associated quadratic approximation is convex everywhere. Conversely, at points of local concavity

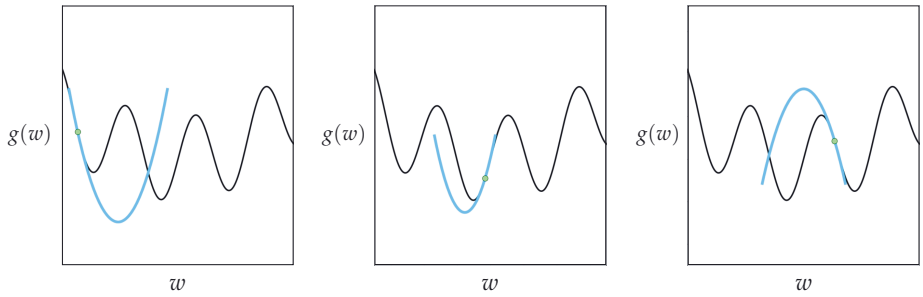

Figure 4.4 Figure associated with Example 4.3. See text for details.

(as in the third panel of the figure) the associated quadratic approximation is universally concave.

4.3 Newton's Method

Since the first-order Taylor series approximation to a function leads to the local optimization framework of gradient descent (see Section 3.5), it seems intuitive that higher-order Taylor series approximations might similarly yield descent-based algorithms as well. In this section we introduce a local optimization scheme based on the second-order Taylor series approximation, called *Newton's method* (named after its creator, Isaac Newton). Because it uses second derivative information, Newton's method has natural strengths and weaknesses when compared to gradient descent. In summary we will see that the cumulative effect of these trade-offs is, in general, that Newton's method is especially useful for minimizing *convex* functions of a moderate number of inputs.

4.3.1 The descent direction

We saw in our discussion of gradient descent that the first-order Taylor series approximation, being a hyperplane itself, conveniently provides us with a descent direction (see Section 3.3). By comparison a quadratic function has *stationary points* that are global minima when the quadratic is convex, and global maxima when it is concave. We can compute the stationary point(s) of a quadratic function fairly easily using the first-order condition for optimality (see Section 3.2).

For the single-input case, the second-order Taylor series approximation centered at a point v is shown in Equation (4.4). Using the first-order condition to solve for the stationary point $w^\star$ of this quadratic (see Example 3.2) by setting its derivative to zero and solving, we find that

$$w^\star = v - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g(v)}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)}. \tag{4.7}$$

Equation (4.7) says that in order to get to the point $w^\star$ we move from v in the *direction* given by $-\frac{\frac{\mathrm{d}}{\mathrm{d}w}g(v)}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(v)}$.

The same kind of calculation can be made in the case of multi-input second-order Taylor series approximation shown in Equation (4.5). Setting the gradient of the quadratic approximation to zero (as shown in Example 3.4) and solving gives the stationary point

$$\mathbf{w}^\star = \mathbf{v} - \left(\nabla^2 g(\mathbf{v})\right)^{-1} \nabla g(\mathbf{v}). \tag{4.8}$$

This is the direct analog of the single-input solution in Equation (4.7), and indeed reduces to it when $N = 1$. It likewise says that in order to get to the stationary point $\mathbf{w}^\star$ we move from $\mathbf{v}$ in the direction given by $-\left(\nabla^2 g(\mathbf{v})\right)^{-1} \nabla g(\mathbf{v})$. When might this direction be a descent direction? Let us examine a simple example first to build up our intuition.

Example 4.4 Stationary points of approximating quadratics

In the top row of Figure 4.5 we show the convex function

$$g(w) = \frac{1}{50}\left(w^4 + w^2\right) + 0.5 \tag{4.9}$$

drawn in black, along with three second-order Taylor series approximations shown in light blue (one per panel), each centered at a distinct input point. In each panel the point of expansion is shown as a red circle and its evaluation by the function as a red $\mathbf{x}$, the stationary point $w^\star$ of the second-order Taylor series as a green circle, and the evaluations of both the quadratic approximation and the function itself at $w^\star$ are denoted by a blue and green $\mathbf{x}$, respectively.

Since the function g itself is convex everywhere, the quadratic approximation not only matches the curvature at each point but is always convex and facing upwards. Therefore its stationary point is always a global minimum. Notice importantly that the minimum of the quadratic approximation $w^\star$ always leads to a lower point on the function than the evaluation of the function at v, i.e., $g\left(w^\star\right) < g\left(v\right)$.

In the bottom row of Figure 4.5 we show similar panels as those described above, only this time for the nonconvex function

$$g(w) = \sin\left(3w\right) + 0.1w^2 + 1.5. \tag{4.10}$$

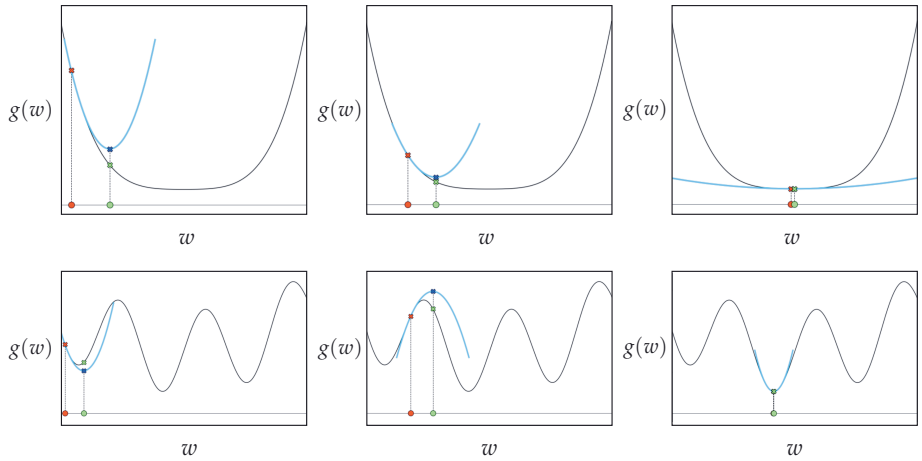

Figure 4.5 Figure associated with Example 4.4. See text for details.

However, the situation is now clearly different, with nonconvexity being the culprit. In particular at concave portions of the function (like the one shown in the middle panel) since the quadratic is also concave, the stationary point $w^\star$ of the quadratic approximation is a global maximum of the approximator, and tends to lead towards points that *increase* the value of the function (not *decrease* it).

From our cursory investigation of these two simple examples we can intuit an idea for a local optimization scheme: repeatedly traveling to points defined by the stationary point of the second-order Taylor series approximation. For convex functions, where each quadratic approximation's stationary point seems to lower the original function's initial evaluation, this idea could provide an efficient algorithm to minimize a cost function. This is indeed the case, and the resulting algorithm is called the *Newton's method*.

4.3.2 The algorithm

Newton's method is a local optimization algorithm produced by repeatedly taking steps to stationary points of the second-order Taylor series approximations of a function. At the kth step of this process for a single-input function, we make a second-order Taylor series approximation centered at the point w^{k-1}

$$h(w) = g(w^{k-1}) + \left(\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})\right)(w - w^{k-1}) + \frac{1}{2}\left(\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1})\right)(w - w^{k-1})^2 \quad (4.11)$$

and solve for its stationary point to create the update w^k as

$$w^k = w^{k-1} - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1})}. \tag{4.12}$$

More generally with multi-input functions taking in N-dimensional input, at the kth step we form the second-order quadratic approximation

$$h(\mathbf{w}) = g(\mathbf{w}^{k-1}) + \nabla g(\mathbf{w}^{k-1})^T(\mathbf{w}-\mathbf{w}^{k-1}) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^{k-1})^T\nabla^2 g\left(\mathbf{w}^{k-1}\right)(\mathbf{w}-\mathbf{w}^{k-1}) \tag{4.13}$$

and solve for a stationary point of this approximator, giving the update $\mathbf{w}^k$ as[1]

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left(\nabla^2 g(\mathbf{w}^{k-1})\right)^{-1}\nabla g(\mathbf{w}^{k-1}). \tag{4.15}$$

This is a local optimization scheme that fits right in with the general form we have seen in the previous two chapters, i.e.,

$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha\mathbf{d}^k \tag{4.16}$$

where, in the case of Newton's method, $\mathbf{d}^k = -\left(\nabla^2 g(\mathbf{w}^{k-1})\right)^{-1}\nabla g(\mathbf{w}^{k-1})$ and $\alpha = 1$. The fact that the steplength parameter α is implicitly set to 1 here follows naturally from the derivation we have seen.

Notice, the Newton's update formula in Equation (4.15) requires that we invert an $N \times N$ Hessian matrix (where N is the input dimension). However, in practice, $\mathbf{w}^k$ is typically found via solving[2] the equivalent symmetric system of equations

$$\nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w} = \nabla^2 g\left(\mathbf{w}^{k-1}\right)\mathbf{w}^{k-1} - \nabla g(\mathbf{w}^{k-1}) \tag{4.17}$$

which can be done more cost-effectively compared to finding its closed form solution via Equation (4.15).

[1] From the perspective of first-order optimization the kth Netwon's method step in Equation (4.12) applied to a single-input function can also be considered a gradient descent step with self-adjusting steplength parameter

$$\alpha = \frac{1}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g\left(w^{k-1}\right)}, \tag{4.14}$$

which adjusts the length traveled based on the underlying curvature of the function, akin to the self-adjusting steplength perspective of normalized gradient steps discussed in Appendix Section A.3. Although this interpretation does not generalize directly to the multi-input case in Equation (4.15), by discarding the off-diagonal entries of the Hessian matrix one can form a generalization of this concept for the multi-input case. See Appendix Section A.8.1 for further details.

[2] One can solve this system using coordinate descent as outlined in Section 3.2.2. When more than one solution exists the smallest possible solution (e.g., in the ℓ_2 sense) is typically taken. This is also referred to as the *pseudo-inverse* of $\nabla^2 g(\mathbf{w})$.

As illustrated in the top panel of Figure 4.6 for a single-input function, starting at an initial point w^0 Newton's method produces a sequence of points w^1, w^2, ..., etc., that minimize g by repeatedly creating the second-order Taylor series quadratic approximation to the function, and traveling to a stationary point of this quadratic. Because Newton's method uses quadratic as opposed to linear approximations at each step, with a quadratic more closely mimicking the associated function, it is often much more effective than gradient descent in the sense that it requires far fewer steps for convergence [14, 15]. However, this reliance on quadratic information also makes Newton's method naturally more difficult to use with nonconvex functions since at concave portions of such a function the algorithm can climb to a local maximum, as illustrated in the bottom panel of Figure 4.6, or oscillate out of control.

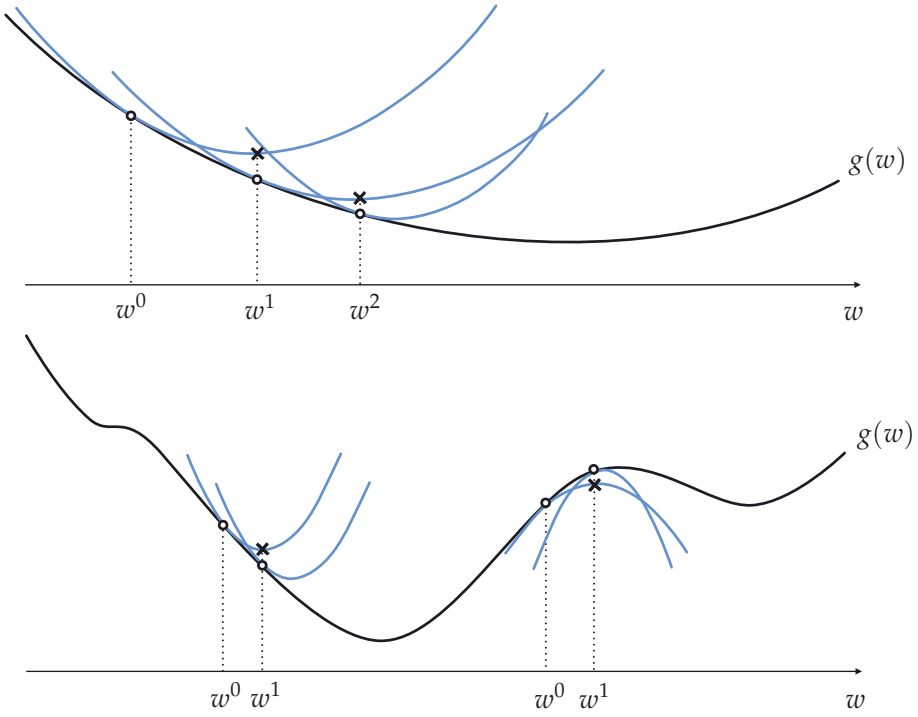

Figure 4.6 Newton's method illustrated. To find a minimum of g, Newton's method hops down the stationary points of quadratic approximations generated by its second-order Taylor series. (top panel) For convex functions these quadratic approximations are themselves always convex (whose only stationary points are minima), and the sequence leads to a minimum of the original function. (bottom panel) For nonconvex functions quadratic approximations can be concave or convex depending on where they are constructed, leading the algorithm to possibly converge to a maximum.

Example 4.5 Minimization of a convex function using Newton's method
In Figure 4.7 we show the process of performing Newton's method to minimize the function

$$g(w) = \frac{1}{50}\left(w^4 + w^2 + 10w\right) + 0.5 \tag{4.18}$$

beginning at the point $w^0 = 2.5$, marked as a green dot in the top-left panel and corresponding evaluation of the function marked as a green x. The top-right panel of the figure shows the first Newton step, with the corresponding quadratic approximation shown in green and its minimum shown as a magenta circle along with the evaluation of this minimum on the quadratic shown as a blue x. The remaining panels show the next iterations of Newton's method.

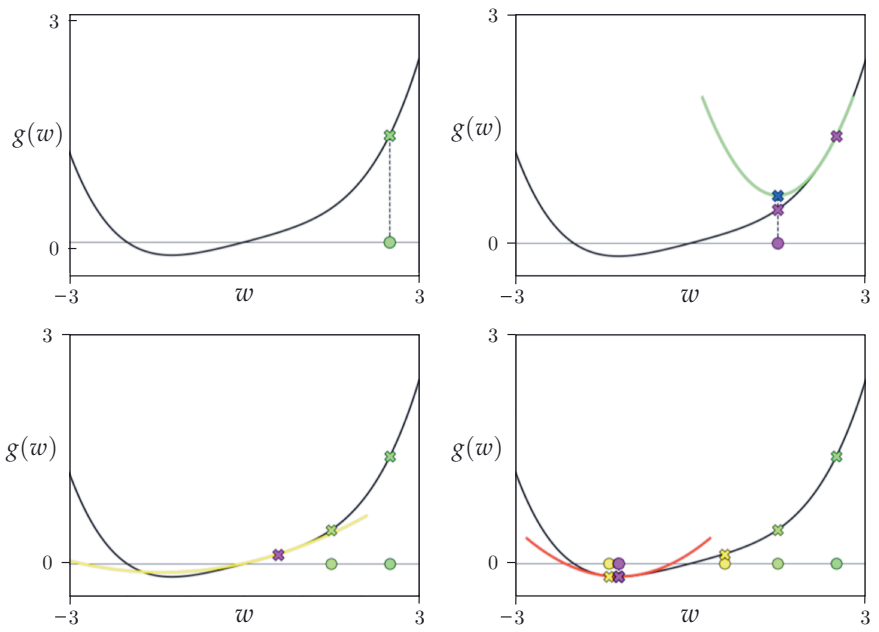

Figure 4.7 Figure associated with Example 4.5, animating a run of Newton's method applied to the function in Equation (4.18). See text for further details.

Example 4.6 Comparison to gradient descent
As illustrated in the right panel of Figure 4.8, a single Newton step is all that is required to completely minimize the convex quadratic function

$$g\left(w_1, w_2\right) = 0.26\left(w_1^2 + w_2^2\right) - 0.48\, w_1 w_2. \tag{4.19}$$

This can be done with a single step because the second-order Taylor series approximation to a quadratic function is simply the quadratic function itself. Thus Newton's method reduces to solving the linear first-order system of a quadratic function. We compare the result of this single Newton step (shown in the right panel of Figure 4.8) to a corresponding run of 100 steps of gradient descent in the left panel of the figure.

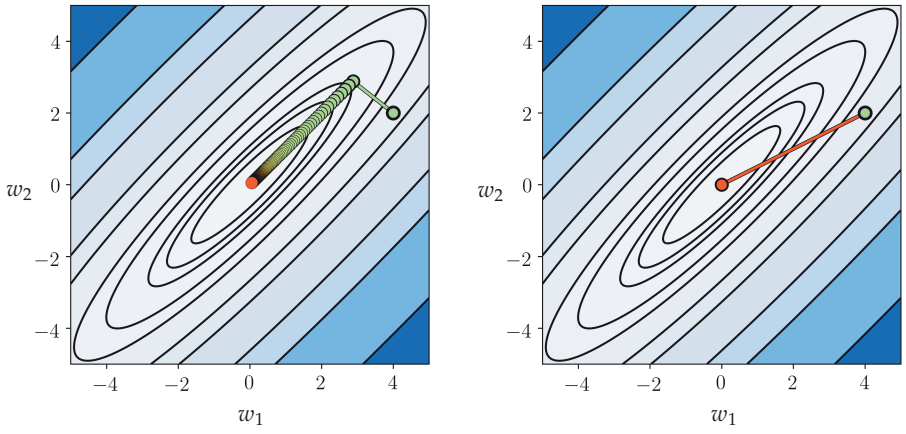

Figure 4.8 Figure associated with Example 4.6. See text for details.

4.3.3 Ensuring numerical stability

Near flat portions of a function the numerator $\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})$ and denominator $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1})$ of the single-input Newton update in Equation (4.12) both have small, near-zero values. This can cause serious numerical problems once each (but especially the denominator) shrinks below *machine precision*, i.e., the smallest value a computer can interpret as being nonzero.

One simple and common way to avoid this potential *division-by-zero* problem is to add a small positive value ϵ to the denominator, either when it shrinks below a certain value or for all iterations. This *regularized* Newton's step then takes the form

$$w^k = w^{k-1} - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g(w^{k-1})}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w^{k-1}) + \epsilon}. \tag{4.20}$$

The value of the *regularization parameter* ϵ is typically set to a small positive[3] value (e.g., 10^{-7}).

[3] This adjustment is made when the function being minimized is known to be convex, since in this case $\frac{\mathrm{d}^2}{\mathrm{d}w^2}g(w) \geq 0$ for all w.

The analogous adjustment for the general multi-input Newtons's update is to add $\epsilon \mathbf{I}_{N\times N}$ (an $N \times N$ identity matrix scaled by a small positive ϵ value) to the Hessian matrix in Equation (4.15), giving[4]

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)^{-1} \nabla g(\mathbf{w}^{k-1}). \tag{4.21}$$

Adding this additional term to the Hessian guarantees that the matrix $\nabla^2 g(\mathbf{w}^{k-1})+ \epsilon \mathbf{I}_{N\times N}$ is always invertible, provided a large enough value for ϵ is used.

4.3.4 Steplength choices

While we have seen in the derivation of Newton's method that (being a local optimization approach) it does have a steplength parameter α, it is implicitly set to $\alpha = 1$ and so appears "invisible." However, in principle, one can explicitly introduce a steplength parameter α and use adjustable methods (e.g., backtracking line search as introduced in Section A.4) to tune it. An explicitly weighted Newton step then takes the form

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \left(\nabla^2 g(\mathbf{w}^{k-1})\right)^{-1} \nabla g(\mathbf{w}^{k-1}) \tag{4.22}$$

with the standard Newton step falling out when $\alpha = 1$.

4.3.5 Newton's method as a zero-finding algorithm

Newton's method was first invented not as a local optimization algorithm, but as a *zero-finding* algorithm. In other words, Newton's method was first invented to find zeros of a function f, i.e., where $f(\mathbf{w}) = \mathbf{0}_{N\times 1}$. Traditionally the function f examined was some sort of polynomial function. In the context of local optimization we can think of Newton's method as an approach for *iteratively* solving the first-order system (see Section 3.2)

$$\nabla g(\mathbf{v}) = \mathbf{0}_{N\times 1}. \tag{4.23}$$

Take the case where our input dimension $N = 1$. Generally speaking, finding zeros of an arbitrary function is not a trivial affair. Instead of trying to solve the first-order equation $\frac{\mathrm{d}}{\mathrm{d}w} g(v) = 0$ directly let us try to set up an iterative procedure where we find an approximate a solution to this equation by solving a related sequence of simpler problems. Following the same sort of logic we used in deriving gradient descent and Newton's method previously, instead of trying to find a zero of the function itself, let us try to find a zero of the tangent line provided by our function's *first-order Taylor series* approximation. Finding

[4] As with the original Newton step in Equation (4.15), it is virtually always more numerically efficient to compute this update by solving the associated linear system $\left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)\mathbf{w} = \left(\nabla^2 g(\mathbf{w}^{k-1}) + \epsilon \mathbf{I}_{N\times N}\right)\mathbf{w}^{k-1} - \nabla g(\mathbf{w}^{k-1})$ for $\mathbf{w}$.

the point(s) at which a line or – more generally – a hyperplane equals zero is a comparatively trivial affair.

To write out the first step of this scheme, remember first and foremost that we are thinking of this as an iterative method applied to the derivative function $\frac{\mathrm{d}}{\mathrm{d}w}g\left(w\right)$. This means that, beginning at a point w^0, our linear first-order Taylor series approximation to the derivative function

$$h\left(w\right) = \frac{\mathrm{d}}{\mathrm{d}w}g\left(w^0\right) + \frac{\mathrm{d}^2}{\mathrm{d}w^2}g\left(w^0\right)\left(w - w^0\right) \tag{4.24}$$

naturally involves the second derivative of the function g (it is, after all, the first-order approximation of this function's derivative). We can easily compute where this line crosses the input axis by setting the equation above equal to zero and solving. Doing this, and calling the solution w^1, we have

$$w^1 = w^0 - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g\left(w^0\right)}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g\left(w^0\right)}. \tag{4.25}$$

Examined closely we can see that this is indeed a Newton step. Since we have only found the zero of a linear approximation to $\frac{\mathrm{d}}{\mathrm{d}w}g\left(w\right)$ and not to this function itself, it is natural to repeat this procedure to refine our approximation. At the kth such step our update takes the form

$$w^k = w^{k-1} - \frac{\frac{\mathrm{d}}{\mathrm{d}w}g\left(w^{k-1}\right)}{\frac{\mathrm{d}^2}{\mathrm{d}w^2}g\left(w^{k-1}\right)} \tag{4.26}$$

which is exactly the form of Newton step in Equation (4.12). Precisely the analogous reasoning applied to multi-input functions (where $N > 1$), starting with the desire to iteratively solve the first-order system, leads to deriving the multi-input Newton's step shown in Equation (4.15).

4.3.6 `Python` implementation

In this section we provide a simple implementation of Newton's method in `Python`, leveraging the excellent `autograd` automatic differentiation and `NumPy` libraries (see Sections 3.4 and B.10). In particular we employ the `grad` and `hessian` modules from `autograd` to compute the first and second derivatives of a general input function automatically.

```
# import autograd's automatic differentiator
from autograd import grad
from autograd import hessian

# import NumPy library
import numpy as np

```

```
# Newton's method
def newtons_method(g, max_its, w, **kwargs):

    # compute gradient/Hessian using autograd
    gradient = grad(g)
    hess = hessian(g)

    # set numerical stability parameter
    epsilon = 10**(-7)
    if 'epsilon' in kwargs:
        epsilon = kwargs['epsilon']

    # run the Newton's method loop
    weight_history = [w]  # container for weight history
    cost_history = [g(w)] # container for cost function history
    for k in range(max_its):

        # evaluate the gradient and hessian
        grad_eval = gradient(w)
        hess_eval = hess(w)

        # reshape hessian to square matrix
        hess_eval.shape = (int((np.size(hess_eval))**(0.5)),int((np.
            size(hess_eval))**(0.5)))

        # solve second-order system for weight update
        A = hess_eval + epsilon*np.eye(w.size)
        b = grad_eval
        w = np.linalg.solve(A, np.dot(A,w)-b)

        # record weight and cost
        weight_history.append(w)
        cost_history.append(g(w))

    return weight_history,cost_history
```

Notice, while we used a maximum iterations convergence criterion the potentially high computational cost of each Newton step often incentivizes the use of more formal convergence criteria (e.g., halting when the norm of the gradient falls below a pre-defined threshold). This also often incentivizes the inclusion of checkpoints that measure and/or adjust the progress of a Newton's method run in order to avoid problems near flat areas of a function. Additionally, one can use the same kind of initialization for this implementation of Newton's method as described for gradient descent in Section 3.5.4.

4.4 Two Natural Weaknesses of Newton's Method

Newton's method is a powerful algorithm that makes enormous progress towards finding a function's minimum at each step, compared to zero- and first-order methods that can require a large number of steps to make equivalent

progress. Since both first- and second-order (i.e., curvature) information are employed, Newton's method does not suffer from the problems inherent to first-order methods (e.g., the zig-zagging problem we saw in Section 3.6.3). However, Newton's method suffers from its own unique weaknesses – primarily in dealing with *nonconvexity*, as well as *scaling* with input dimension – which we briefly discuss here. While these weaknesses do not prevent Newton's method (as we have described it) from being widely used in machine learning, they are (at least) worth being aware of.

4.4.1 Minimizing nonconvex functions

As discussed in the previous section, Newton's method can behave very badly when applied to minimizing nonconvex functions. Since each step is based on the second-order approximation to a function, initiated at *concave* point/region Newton's method will naturally take a step *uphill*. This fact is illustrated for a prototypical nonconvex function in the bottom row of Figure 4.6. The interested reader can see Section A.7, where we describe a simple and common approach to adjusting Newton's method to address this issue.

4.4.2 Scaling limitations

Since the quadratic approximation used by Newton's method matches a function very well *locally*, the method can converge to a global minimum in far fewer steps (than first-order methods) particularly when close to a minimum. However, a Newton's method step is computationally far more expensive than a first-order step, requiring the storage and computation of not just a gradient but an entire $N \times N$ Hessian matrix of second derivative information as well. Simply storing the Hessian for a single step of Newton's method, with its N^2 entries, can quickly become challenging for even moderately sized input. For example, if the input to a function has dimension $N = 10,000$ the corresponding Hessian matrix has $100,000,000$ entries. The kind of functions used in machine learning applications can easily have tens of thousands to hundreds of thousands or even millions of inputs, making the complete storage of an associated Hessian impossible.

Later in Section A.8 we discuss basic ways of ameliorating this problem, which involve adjusting the basic Newton's method step by replacing the Hessian with some sort of approximation that does not suffer from this inherent scaling issue.

4.5 Conclusion

In this chapter we cap off our discussion of mathematical optimization in this part of the text by describing second-order optimization techniques, i.e., those that leverage both the first and second derivative(s) of a function in forming descent directions.

We began in Section 4.2 with a review of the second-order condition for optimality. We then briefly touched on function curvature as defined by its second derivative(s) in Section 4.1 before immediately applying this concept in detailing the keystone second-order local optimization method – *Newton's method* – in Section 4.3.

Afterwards in Section 4.4 we touched on two natural problems with the Newton's method scheme – its application to the minimization of nonconvex functions and to functions with high-dimensional input. The interested reader should note that in Appendix Sections A.7 and A.8 we detail common adjustments to the standard Newton's scheme for ameliorating these problems, with the latter set of adjustments referred to as *Hessian-free* optimization.

4.6 Exercises

† The data required to complete the following exercises can be downloaded from the text's github repository at `github.com/jermwatt/machine_learning_refined`

4.1 Determining the eigenvalues of a symmetric matrix

In this exercise we investigate an alternative approach to checking that the eigenvalues of an $N \times N$ symmetric matrix $\mathbf{C}$ (e.g., a Hessian matrix) are all nonnegative. This approach does not involve explicitly computing the eigenvalues themselves, and is significantly easier to employ in practice.

(a) Let $\mathbf{C}$ be an $N \times N$ symmetric matrix. Show that if $\mathbf{C}$ has all nonnegative eigenvalues then the quantity $\mathbf{z}^T\mathbf{C}\mathbf{z} \geq 0$ for all $\mathbf{z}$. *Hint: use the eigenvalue decomposition of* $\mathbf{C}$ *(see Appendix Section C.4).*

(b) Show the converse. That is, if an $N \times N$ symmetric matrix $\mathbf{C}$ satisfies $\mathbf{z}^T\mathbf{C}\mathbf{z} \geq 0$ for all $\mathbf{z}$ then it must have all nonnegative eigenvalues.

(c) Use this method to verify that the second-order definition of convexity holds for the quadratic function

$$g(\mathbf{w}) = a + \mathbf{b}^T\mathbf{w} + \mathbf{w}^T\mathbf{C}\mathbf{w} \tag{4.27}$$

where $a = 1$, $\mathbf{b} = [1 \;\; 1]^T$, and $\mathbf{C} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$.

(d) Show that the eigenvalues of $\mathbf{C} + \lambda\mathbf{I}_{N\times N}$ can all be made to be positive by

setting λ large enough. What is the smallest value of λ that will make this happen?

4.2 Outer-product matrices have all nonnegative eigenvalues

(a) Use the method described in Exercise 4.1 to verify that for any $N \times 1$ vector $\mathbf{x}$, the $N \times N$ outer-product matrix $\mathbf{x}\mathbf{x}^T$ has all nonnegative eigenvalues.

(b) Similarly show that for any set of P vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_P$ of length N that the sum of outer-product matrices $\sum_{p=1}^{P} \delta_p \mathbf{x}_p \mathbf{x}_p^T$ has all nonnegative eigenvalues if each $\delta_p \geq 0$.

(c) Show that the matrix $\sum_{p=1}^{P} \delta_p \mathbf{x}_p \mathbf{x}_p^T + \lambda \mathbf{I}_{N\times N}$ where each $\delta_p \geq 0$ and $\lambda > 0$ has all positive eigenvalues.

4.3 An alternative way to check the second-order definition of convexity

Recall that the second-order definition of convexity for a multi-input function $g(\mathbf{w})$ requires that we verify whether or not the eigenvalues of $\nabla^2 g(\mathbf{w})$ are nonnegative for each input $\mathbf{w}$. However, to explicitly compute the eigenvalues of the Hessian in order to check this is a cumbersome or even impossible task for all but the nicest of functions. Here we use the result of Exercise 4.1 to express the second-order definition of convexity in a way that is often much easier to employ in practice.

(a) Use the result of Exercise 4.1 to conclude that nonnegativity of the eigenvalues of $\nabla^2 g(\mathbf{w})$ are nonnegative at every $\mathbf{w}$ is equivalently stated as the inequality $\mathbf{z}^T \left(\nabla^2 g(\mathbf{w})\right) \mathbf{z} \geq 0$ holding at each $\mathbf{w}$ for all $\mathbf{z}$.

(b) Use this manner of expressing the second-order definition of convexity to verify that the general quadratic function $g(\mathbf{w}) = a + \mathbf{b}^T\mathbf{w} + \mathbf{w}^T\mathbf{C}\mathbf{w}$ where $\mathbf{C}$ is symmetric and known to have all nonnegative eigenvalues, always defines a convex function.

(c) Verify that $g(\mathbf{w}) = -\cos\left(2\pi\mathbf{w}^T\mathbf{w}\right) + \mathbf{w}^T\mathbf{w}$ is nonconvex by showing that it does *not* satisfy the second-order definition of convexity.

4.4 Newton's method I

Repeat the experiment described in Example 4.5. Instead of plotting the re-

sulting path taken by Newton's method (as shown in Figure 4.7), create a cost function history plot to ensure your algorithm properly converges to a point near the global minimum of the function. You may employ the implementation of Newton's method described in Section 4.3.6 as a base for this exercise.

4.5 Newton's method II

(a) Use the first-order optimality condition (see Section 3.2) to determine the unique stationary point of the function $g(\mathbf{w}) = \log\left(1 + e^{\mathbf{w}^T\mathbf{w}}\right)$ where $\mathbf{w}$ is two-dimensional (i.e., $N = 2$).

(b) Use the second-order definition of convexity to verify that $g(\mathbf{w})$ is convex, implying that the stationary point found in part (a) is a global minimum. *Hint: to check the second-order definition use Exercise 4.2.*

(c) Perform Newton's method to find the minimum of the function $g(\mathbf{w})$ determined in part (a). Initialize your algorithm at $\mathbf{w}^0 = \mathbf{1}_{N\times 1}$ and make a plot of the cost function history for ten iterations of Newton's method in order to verify that your algorithm works properly and is converging. You may use the implementation given in Section 4.3.6 as a base for this part of the exercise.

(d) Now run your Newton's method code from part (c) again, this time initializing at the point $\mathbf{w}^0 = 4 \cdot \mathbf{1}_{N\times 1}$. While this initialization is further away from the unique minimum of $g(\mathbf{w})$ than the one used in part (c) your Newton's method algorithm should converge *faster* starting at this point. At first glance this result seems very counterintuitive, as we (rightfully) expect that an initial point closer to a minimum will provoke more rapid convergence of Newton's method! Explain why this result actually makes sense for the particular function $g(\mathbf{w})$ we are minimizing here.

4.6 Finding square roots

Use Newton's method to compute the square root of 999. Briefly explain how you set up the relevant cost function that was minimized to obtain this square root. Explain how you use zero- or first-order optimization methods (detailed in Chapters 2 and 3) to do this as well.

4.7 Nonconvex minimization using Newton's method

Use (regularized) Newton's method to minimize the function

$$g(w) = \cos(w) \tag{4.28}$$

beginning at $w = 0.1$. In particular make sure you achieve decrease in function value at *every* step of Newton's method.

4.8 Newtonian descent

(a) Show that when $g(\mathbf{w})$ is convex the Newton step in Equation (4.15) does indeed decrease the evaluation of g, i.e., $g\left(\mathbf{w}^k\right) \leq g\left(\mathbf{w}^{k-1}\right)$.

(b) Show, regardless of the function g being minimized, that ϵ in Equation (4.21) can be set large enough so that a corresponding Newton step can lead to a lower portion of the function, i.e., $g\left(\mathbf{w}^k\right) \leq g\left(\mathbf{w}^{k-1}\right)$.

4.9 Newton's method as a self-adjusting gradient descent method

Implement the subsampled Newton's step outlined in Appendix Section A.8.1 and given in Equation (A.78) formed by ignoring all off-diagonal elements of the Hessian, and compare it to gradient descent using the test function

$$g(\mathbf{w}) = a + \mathbf{b}^T\mathbf{w} + \mathbf{w}^T\mathbf{C}\mathbf{w} \tag{4.29}$$

where $a = 0$, $\mathbf{b} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, and $\mathbf{C} = \begin{bmatrix} 0.5 & 2 \\ 1 & 9.75 \end{bmatrix}$.

Make a run of each local method for 25 steps beginning at the initial point $\mathbf{w}^0 = [10 \;\; 1]^T$, using the largest fixed steplength value of the form 10^{γ} (where γ is an integer) for gradient descent. Make a contour plot of the test function and plot the steps from each run on top of it to visualize how each algorithm performs.

4.10 The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method

Start with the same assumption as in Example A.12 (i.e., a recursion based on a rank-2 difference between $\mathbf{S}^k$ and its predecessor) and employ the secant condition to derive a recursive update for $\mathbf{S}^k$ in terms of $\mathbf{S}^{k-1}$, $\mathbf{a}^k$, and $\mathbf{b}^k$. Next, use the Sherman–Morrison identity to rewrite your update in terms of $\mathbf{F}^k$, the inverse of $\mathbf{S}^k$.